Introdução ao Spring Data JPA

Site: Moodle institucional da UTFPR
Curso: CETEJ34 - Sistemas de Banco de Dados - JAVA_XXX (2024_01)
Livro: Introdução ao Spring Data JPA

Impresso por: PAULO ROBERTO DOS SANTOS
Data: segunda-feira, 24 jun. 2024, 16:22

Índice

1. Introdução

O Spring Framework é um dos mais populares e poderosos frameworks dentro da linguagem Java.

Ele não se limita a um único recurso, mas sim, a uma larga variedade de funcionalidades que fazem parte do dia a dia de um programador Java no desenvolvimento de aplicações.

Suas funcionalidades mais famosas talvez sejam o seu módulo Web, conhecido por Spring MVC e o núcleo principal do Spring que provê o conceito de injeção de dependências e inversão de controle.

Programar utilizando o Spring Framework não se limita apenas a usar um ou mais de seus recursos, mas também fazer uso das boas práticas e padrões de projetos, os quais aumentam a produtividade, facilitam a manutenção e atualização do código bem como o reuso, princípio básico da orientação a objetos.

Definir o Spring Framework sempre é uma tarefa difícil, pois ele cobre tantos conceitos que é quase impossível dizer especificamente o que ele faz. Você pode definir o Hibernate como um framework ORM e explicar exatamente sua função, ou pode conceituar o Apache Struts 2 com relação ao padrão MVC. Mas definir o Spring Framework realmente não é fácil.

Na documentação, o Spring é citado como um framework leve, projetado para a construção de aplicativos Java. Ser considerado leve tem o sentido referente a você realizar apenas algumas poucas alterações no código de sua aplicação para obter seus benefícios.

Uma característica marcante do Spring Framework é a possibilidade de integração com bibliotecas, especificações Java e diversos frameworks como o Hibernate, Apache Struts, JasperReports, Dandelion, Direct Web Remoting (DWR), Java Server Faces (JSF 2), entre outros.

Um conceito importante sobre o Spring Framework está diretamente ligado à injeção de dependências e inversão de controle. Cada classe que será gerenciada por esse sistema recebe o nome de *bean*. A partir de *beans* você usa diversas classes e interfaces do Spring e também pode criar seus próprios *beans* para serem gerenciados pelo framework e assim, não precisa mais se preocupar em criar a instância de seus objetos sempre que precisar deles. O Spring fará isso por você.

1.1. Spring Data

Persistência com banco de dados é algo bem comum no mundo da programação Web. É praticamente impossível pensar em um projeto que não realize de alguma forma o acesso a uma base de dados.

Inicialmente, no Java, as operações com banco de dados eram realizadas exclusivamente via API JDBC. Com o passar do tempo, surgiu o conceito de mapeamento objeto relacional, conhecido com o ORM. Assim, vários frameworks ORM foram lançados, como o Eclipselink, Toplink, MyBatis, entre outros, e o mais famoso deles, o Hibernate.

Embora o Spring não seja um framework ORM ele proporciona a integração com esses recursos. Se já é fácil trabalhar com persistência de dados usando, por exemplo, o Hibernate, quando ele é integrado ao Spring Framework fica ainda mais simples.

Porém, os desenvolvedores do Spring não estavam satisfeitos e pensaram em algo ainda mais produtivo. Foi então elaborado o projeto Spring Data, baseado nos conceitos de Design Pattern Repository.

O Spring Data pode ser chamado de projeto principal, o qual contém subprojetos tais como: SpringData JPA, Spring Data MongoDB, SpringData REST , Spring Data Redis, entre outros.

O Spring Data é baseado em um grupo principal de interfaces que aparecem em todos seus subprojetos. Entretanto, em alguns desses subprojetos existe uma interface mais específica para o uso de tal recurso, como: **JpaRepository** no Spring Data JPA e **MongoRepository** no Spring Data MongoDB. As interfaces consideradas principais são as: **Repository**, **CrudRepository** e **PagingAndSortingRepository**.

1.2. Spring Data JPA

Embora o Spring Data tenha inúmeros subprojetos, este curso terá como foco apenas o Spring Data JPA, que fornece o acesso a banco de dados relacionais via JPA.

A JPA é uma especificação Java (JSR-317) que foi desenvolvida para padronizar o acesso, métodos de persistência e o gerenciamento de dados entre os objetos Java e bancos de dados relacionais (MySQL, H2, Oracle, Derby DB, DB2, HSQL DB, ...).

Deste modo, a camada de persistência em uma aplicação Java, que seja baseada na especificação, pode ser usada por qualquer framework ORM que seja especificado pela JSR. Isto permite que o código-fonte desta camada não sofra qualquer alteração se houver a necessidade de mudar de framework ORM. Ou seja, se você usa o Eclipselink e em algum momento precisa passar a utilizar o Hibernate, está garantindo que seu código não sofrerá alterações.

Seguindo este mesmo princípio, para trabalhar com o Spring Data JPA é necessário também integrá-lo a um framework ORM que siga a especificação JPA.

Um framework ORM não é nada mais do que uma camada de acesso a dados que implementa a API JDBC. E seu objetivo é fornecer uma espécie de intercâmbio entre um banco de dados relacional e objetos Java. Assim, é possível que o programador trabalhe de forma 100% orientada a objetos, sem que ele precise lidar diretamente em seu código com tabelas, linhas e colunas de um banco de dados. Esta responsabilidade fica por conta do framework ORM que também pode ser chamado de provedor (Provider) quando usado junto ao Spring Framework.

Muitas vezes, palavras não são suficientes para se entender o quanto o Spring Data JPA torna simples o acesso a dados. Então, observe o código da **Listagem 1.1,** uma classe que usa os métodos da especificação JPA para localizar no banco de dados uma lista de pessoas pelo critério de idade.

LISTAGEM 1.1: REPOSITÓRIO COM INSTRUÇÕES **JPA**

```
@Repository
public class PessoaRepository {

   @PersistenceContext
   private EntityManager entityManger;

   public list<Pessoa> findByidade(Integer idade) {

      String query = "from Pessoa p where p.idade = :idade";

      return entityManger
         .createQuery(query,user.class)
         .setParaemter("idade", idade)
         .getResultlist();
   }
}
```

A classe **Pessoa Repository,** do código-fonte apresentado, possui um método chamado **findByidade(),** o qual recebe como parâmetro um inteiro que representa uma idade qualquer selecionada em algum ponto da aplicação. No corpo deste método existe uma variável chamada **query,** a qual é do tipo **String.** Esta variável contém uma consulta JPQL que seleciona todas as pessoas com a idade equivalente ao valor do parâmetro **idade.**

A consulta é processada pelo método **createQuery()** do objeto entityManager. E o parâmetro idade é adicionado ao setParameter() , método responsável por incluir este valor a clausula where da consulta JPQL.

Por fim, getResultList() retorna a lista de pessoas referentes a idade selecionada. Este é o processo básico para realizar um a consulta via JPA. Veja que o código, embora seja simples, exigiu uma determinada quantidade de ações para obter o resultado desejado.

Agora é possível analisar o código fonte da **Listagem1.2.** Ele possui uma consulta com a mesma finalidade do código anterior, porém, desta vez, usando o **Spring Data JPA**:

LISTAGEM 1.2: REPOSITÓRIO COM INSTRUÇÕES VIA **SPRING DATA JPA**

```
public interface PessoaRepository extends JpaRepository<Pessoa,Long> {
   List<Pessoa> findByidade(Integer idade);
}
```

Veja que, ao invés de uma classe, foi utilizada uma interface e ela estende a interface **JpaRepository** do Spring Data JPA . Desta forma, marcamos **PessoaRepository** como uma interface gerenciada pelo Spring.

O método **findByidade()** não tem corpo, já que estamos lidando com uma interface. E onde devemos então, implementar o corpo desde método? Na verdade em lugar algum. Isto mesmo, com o uso do SpringData JPA não precisamos criar o corpo de nossos métodos, o próprio Spring Data faz isso para nós em tempo de execução. Não ficou muito mais simples e rápido do que o código da **Listagem 1.1?** Com certeza ficou.

Com tudo, você deve estar se perguntando: "Como o Spring vai criar o corpo do método em tempo de execução?".

Na verdade, o corpo do método não é desenvolvido pelo Spring Data. O que acontece é que o Spring Data usa um processo baseado em **palavras chaves** para interpretar o que a consulta precisa localizar no banco de dados.

Todos os métodos incluídos em uma interface do tipo **JpaRepository,** deve seguir um modelo de nomes que usa certas palavras-chave *(keywords)* que são processadas pelo Spring Data.

Então, o método **findByIdade()** é analisado mais ou menos da seguinte forma: é dito que temos uma consulta pelas palavras **findBy**. E que esta consulta tem como critério uma idade, pela palavra-chave **Idade.**

Vamos ver mais sobre esta técnica posteriormente, mas basicamente, temos o método de consulta já finalizado e pronto para ser usado a qualquer momento.